**Software Requirement Specification**

**NAME OF SYSTEM : Bakery**

**DATE : 5/9/2024**

Version 1.2

**Logo :**

**Presented To: Dr.Mohamed Ramdan**

**Submitted By: Technical team**

Group names :2ABK

**TABLE OF CONTENTS**

**REVISION HISTORY**

| **Date** | **Author** | **Distributed to** | **Version** | **Description** |
| --- | --- | --- | --- | --- |
| 4/7/2024 | Doc Team | Relevant stakeholders | 1.0 | first version of  documentation  initializing  requirements |
| 5/1/2024 | Doc Team | Relevant stakeholders | 1.1 | updating use cases  checking completed requirement |
| 9/5/2024 | Doc Team | Relevant stakeholders | 1.2 | updating some  diagrams  and  checking completed  requirements |

**1 Introduction**

The project aims to develop a bakery website with a selling system for online orders. The Software Development Plan outlines the project scope, objectives, schedule, resources, and risk management. Its purpose is to guide the development team and ensure alignment with stakeholder expectations for successful project execution.

**1.2 Scope**

### Actors

#### 1. Customer

* **Description**: Represents individuals who visit the bakery website to browse and purchase bakery products.
* **Needs/Goals**: Convenient browsing, easy checkout process, secure payment options, access to product information and reviews.
* **Actions**: Browsing products, adding items to cart, proceeding to checkout, making payments, leaving reviews and feedback.

#### 2.Administrator

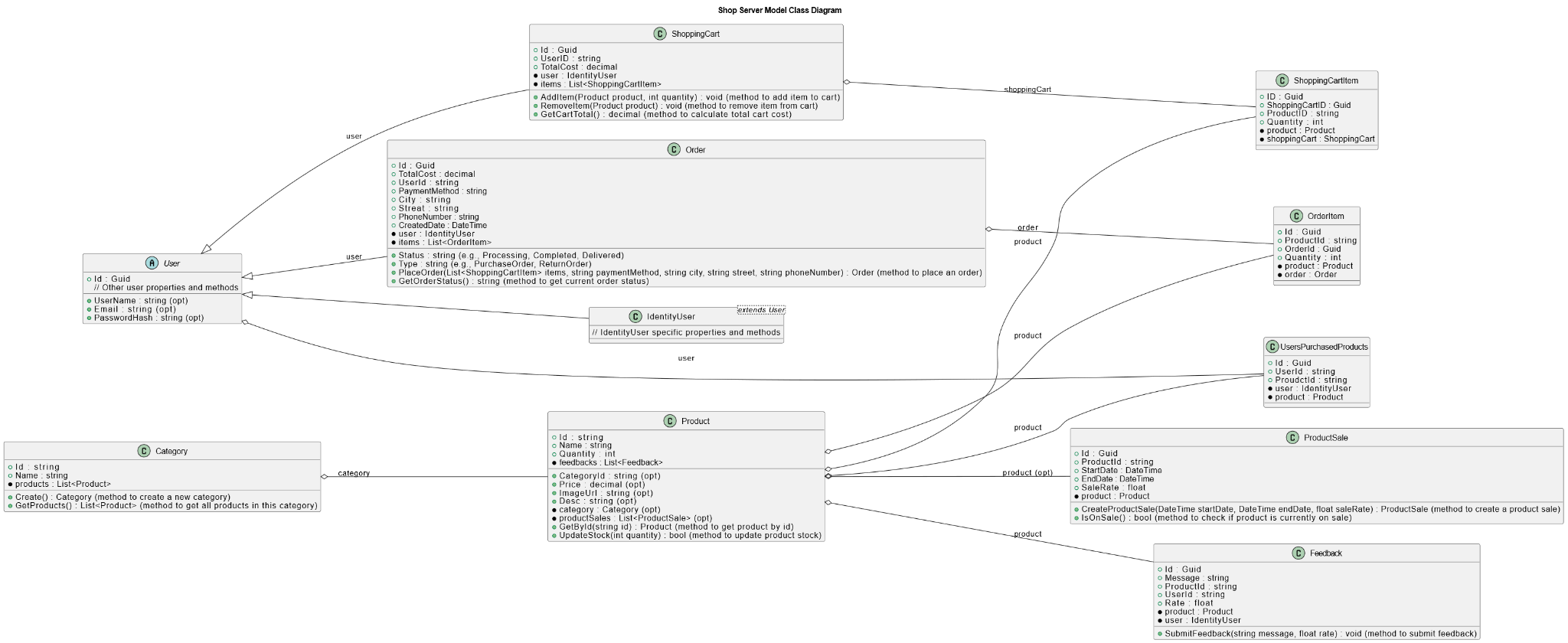
* **Description**: Represents individuals responsible for managing the bakery's online presence, including inventory, orders, and customer interactions.
* **Needs/Goals**: Efficient management of inventory, processing orders, monitoring sales, updating product information, managing customer inquiries.
* **Actions**: Adding/editing/deleting products, managing inventory levels, processing orders, responding to customer inquiries, analyzing sales data.

**2 Related Documents**

Requirements Traceability matrix reference :

| **Requirement ID** | **Description** | **Design Document** | **Implementation** |
| --- | --- | --- | --- |
| REQ-001 | The system shall allow users to create an account. | DD-001 | Implemented |
| REQ-002 | Users shall be able to browse bakery products. | DD-002 | Implemented |
| REQ-003 | The system must support online payments. | DD-003 | NOT Implemented |
| REQ-004 | Customers should receive email notifications for order confirmations. | DD-004 | Not Implemented |
| REQ-005 | Admins should be able to manage inventory. | DD-005 | Implemented |
| REQ-006 | The system should generate sales reports. | DD-006 | Not Implemented |

**Class Diagram:-**

****

**The class diagram consists of the following classes and their properties:**

\* \*\*User:\*\* This abstract class represents a user of the shop system. It includes properties like `Id` (unique identifier), `UserName` (optional), `Email` (optional), and `PasswordHash` (optional) for authentication purposes.

\* \*\*IdentityUser:\*\* This class inherits from `User` and likely contains additional properties and methods specific to the chosen identity framework (e.g., ASP.NET Identity).

\* \*\*Category:\*\* This class represents a product category within the shop. It includes properties like `Id` (unique identifier) and `Name` to categorize products. It also has methods like `Create()` to create a new category and `GetProducts()` to retrieve all products belonging to the category.

\* \*\*Product:\*\* This class represents a product sold in the shop. It includes properties like `Id` (unique identifier), `CategoryId` (optional - foreign key to a category), `Name`, `Price` (optional), `ImageUrl` (optional - URL for product image), `Desc` (optional - product description), and `Quantity` (current stock level). It also has methods like `GetById(string id)` to retrieve a product by its ID and `UpdateStock(int quantity)` to update the product's stock quantity.

\* \*\*Feedback:\*\* This class represents feedback submitted by a user for a product. It includes properties like `Id` (unique identifier), `Message` (feedback content), `ProductId` (foreign key to the product), `UserId` (foreign key to the user who submitted the feedback), and `Rate` (rating given to the product). It also has a method `SubmitFeedback(string message, float rate)` to submit new feedback.

\* \*\*Order:\*\* This class represents an order placed by a user. It includes properties like `Id` (unique identifier), `TotalCost`, `UserId` (foreign key to the user who placed the order), `Status` (current order status - e.g., Processing, Completed, Delivered), `Type` (order type - e.g., PurchaseOrder, ReturnOrder), `PaymentMethod`, `City`, `Streat` (address details), `PhoneNumber`, and `CreatedDate` (date and time the order was placed). It also has methods like `PlaceOrder(List<ShoppingCartItem> items, string paymentMethod, string city, string street, string phoneNumber)` to place a new order and `GetOrderStatus()` to retrieve the current order status.

\* \*\*OrderItem:\*\* This class represents an item within an order. It includes properties like `Id` (unique identifier), `ProductId` (foreign key to the product in the order), `OrderId` (foreign key to the order), and `Quantity` (number of units of the product ordered).

\* \*\*ProductSale:\*\* This class represents a promotional sale applied to a product. It includes properties like `Id` (unique identifier), `ProductId` (foreign key to the product on sale), `StartDate` (start date of the sale), `EndDate` (end date of the sale), and `SaleRate` (discount percentage). It also has methods like `CreateProductSale(DateTime startDate, DateTime endDate, float saleRate)` to create a new product sale and `IsOnSale()` to check if the product is currently on sale.

\* \*\*ShoppingCart:\*\* This class represents a user's shopping cart. It includes properties like `Id` (unique identifier), `UserID` (foreign key to the user), and `TotalCost` (total cost of items in the cart). It also has methods like `AddItem(Product product, int quantity)` to add an item to the cart, `RemoveItem(Product product)` to remove an item from the cart, and `GetCartTotal()` to calculate the total cost of items in the cart.

\* \*\*ShoppingCartItem:\*\* This class represents an item within a shopping cart. It includes properties like `ID` (unique identifier), `ShoppingCartID` (foreign key to the shopping cart), `ProductID` (foreign key to the product in the cart), and `Quantity` (number of units of the product added to the cart).

\* \*\*UsersPurchasedProducts:\*\* This class likely serves for tracking user purchase history. It includes properties like `Id` (unique identifier), `UserId` (foreign key to the user), and `ProudctId` (foreign key to the purchased product).

### Relationships

The class diagram depicts the following relationships between the classes:

\* \*\*Inheritance:\*\* `IdentityUser` inherits from `User`.

\* \*\*Aggregation:\*\*

\* A `Category` can have many `Product` objects associated with it (one-to-many).

**Requirements**

**Functional requirements:-**

**Customer:**

* browse through various bakery products.
* As a customer, I want to create an account to facilitate easier checkout for future orders.
* As a customer, I want to be able to add items to my shopping cart.
* As a customer, I want to view detailed information about each bakery product, including ingredients, price, and availability.
* As a customer, I want to be able to place an order and choose a convenient delivery or pickup option.

**Administrator:**

* As an administrator, I want to manage the bakery's product inventory, including adding new products, updating existing ones, and removing discontinued items.
* As an administrator, I want to manage user accounts, including creating new accounts, updating user information, and deactivating accounts if necessary.

**Non-Functional Requirements**

**Security**

**Description:** The bakery website must prioritize security measures to protect user data and ensure safe transactions.

* **Measurable Requirement:** The website must utilize SSL (Secure Sockets Layer) certification to encrypt data transmissions, providing a secure browsing experience for users.

**Performance**

**Description:** The bakery website must deliver consistent performance to meet user expectations and maintain satisfaction.

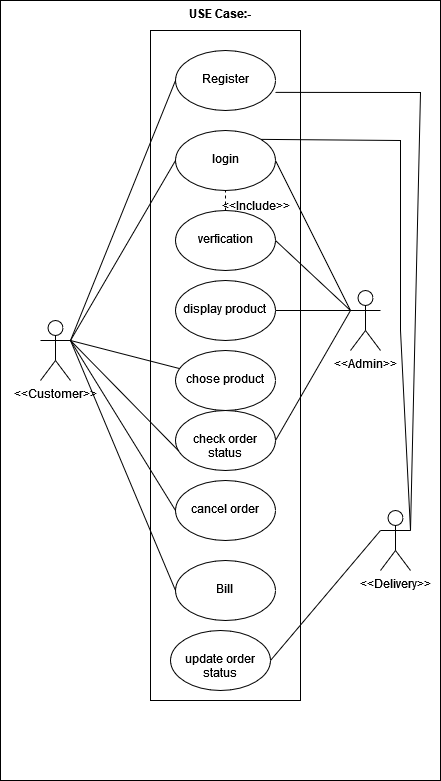
* **Measurable Requirement:**
  + **Uptime:** The website must maintain at least 99% uptime, ensuring minimal downtime for users to access services.
  + **Response Time:** The average page load time should not exceed [specified time], ensuring a responsive browsing experience for users across different devices and network conditions.

**Maintainability**

**Description:** The bakery website must be designed and developed in a way that facilitates easy maintenance and updates.

* **Measurable Requirement:**
  + **Regular Updates:** The website must undergo regular maintenance updates to address security vulnerabilities, optimize performance, and introduce new features or improvements.
  + **Documentation:** Comprehensive documentation outlining the website's architecture, codebase, and operational procedures must be maintained to aid future development and troubleshooting efforts.

**4.1Use Case diagram :-**



**4.2 Use Cases**

**4.2.1 Use Case Description**

**Description: This section provides a detailed description of a specific use case within the bakery website system.**

**Use Case: Browse Bakery Products**

**Name: BrowseProducts**

**Participating Actor(s): Customer**

**Entry: The customer accesses the bakery website.**

**Exit: The customer finds the desired bakery product or completes the browsing session.**

**Flow of Events:**

1. **The customer opens the bakery website in their web browser.**
2. **The system displays the homepage, showcasing featured bakery products and promotions.**
3. **The customer navigates to the product catalog section.**
4. **The system retrieves and displays a list of available bakery products, including images, names, prices, and brief descriptions.**
5. **The customer selects a specific product to view more detailed information.**
6. **The system presents the product details page, showing additional information such as ingredients, nutritional facts, customer reviews, and related products.**
7. **The customer can add the selected product to their shopping cart or continue browsing.**
8. **If the customer adds the product to their cart, the system updates the cart summary to reflect the added item.**
9. **The customer may choose to proceed to checkout or continue browsing for more products.**
10. **If the customer decides to check out, the system redirects them to the checkout process.**

**Special Requirements:**

* **The system should load product images and information quickly to provide a seamless browsing experience.**
* **The product catalog should be easily navigable, allowing customers to find desired items efficiently.**

**Use Case: Manage Product Inventory**

**Name: ManageInventory**

**Participating Actor(s): Administrator**

**Entry: The administrator logs into the bakery website admin dashboard.**

**Exit: The administrator successfully manages the bakery's product inventory.**

**Flow of Events:**

1. **The administrator accesses the admin dashboard after logging in.**
2. **The system presents various administrative options, including inventory management.**
3. **The administrator selects the inventory management option.**
4. **The system displays a list of current bakery products along with their details such as name, quantity, and status.**
5. **The administrator can add new products to the inventory by providing necessary information such as name, description, price, and quantity.**
6. **The administrator can update existing product information, including price, quantity, and status (e.g., available, out of stock).**
7. **The administrator can remove discontinued or outdated products from the inventory.**
8. **The system updates the inventory database with any changes made by the administrator.**

**Special Requirements:**

* **The admin dashboard should provide a user-friendly interface for easy navigation and efficient inventory management.**
* **Changes to the inventory should be reflected immediately in the system to ensure accurate stock levels.**

**Use Case: Process Customer Orders**

**Name: ProcessOrders**

**Participating Actor(s):System**

**Entry: The administrator accesses the admin dashboard.**

**Exit: The administrator successfully processes customer orders.**

**Flow of Events:**

1. **The administrator logs into the admin dashboard.**
2. **The system presents options for order management.**
3. **The administrator selects the order management option.**
4. **The system displays a list of pending customer orders, including order details such as customer name, order items, and total amount.**
5. **The administrator reviews each order and verifies the availability of products in the inventory.**
6. **If all products are available, the administrator confirms the order and proceeds to generate an invoice.**
7. **The system updates the order status to "confirmed" and generates an invoice for the customer.**
8. **The administrator notifies the customer of the order confirmation and provides details regarding delivery or pickup options.**